Group-based Replication of On-line Transaction Processing Servers[⋆]

A. Correia Jr. A. Sousa L. Soares J. Pereira F. Moura R. Oliveira

Computer Science and Technology Center
Computer Science Department
University of Minho

Abstract. Several techniques for database replication using group communication have recently been proposed, namely, the Database State Machine, Postgres-R, and the NODO protocol. Although all rely on a totally ordered multicast for consistency, they differ substantially on how multicast is used. This results in different performance trade-offs which are hard to compare as each protocol is presented using a different load scenario and evaluation method.
In this paper we evaluate the suitability of such protocols for replication of On-Line Transaction Processing (OLTP) applications in clusters of servers and over wide area networks. This is achieved by implementing them using a common infra-structure and by using a standard workload. The results allows us to select the best protocol regarding performance and scalability in a demanding but realistic usage scenario.

1 Introduction

Synchronous database replication provides both transparent distribution and fault-tolerance. By keeping data strictly up-to-date in all replicas, application programmers do not have to manage complex reconciliation procedures and fail-over can happen without causing any committed updates to be lost. Recently, replication techniques based on group communication have been proposed as a means to overcome performance bottlenecks and make synchronous replication cost-effective [1, 12, 11, 17, 15, 20].

In contrast with replication based on distributed locking and atomic commit protocols, group communication based protocols minimize interaction between replicas and the resulting synchronization overhead by relying on total order multicast to ensure consistency. Generically, the approach builds on the classical replicated state machine [19]: The exact same sequence of update operations is applied to the same initial state, thus producing a consistent replicated output and final state. The problem is then to ensure deterministic processing without overly restricting concurrent execution, which would dramatically reduce throughput, and avoid re-execution in all replicas.

These concerns have been addressed by several proposals based on group communication [14, 15, 17, 13]. Although all rely on a totally ordered multicast for consistency, they differ mainly in whether transactions are executed conservatively [14, 15] or optimistically [17, 13]. In the former, by a priori coordination among the replicas, it is

[⋆] Supported by FCT, project STRONGREP (POSI/CHS/41285/2001).

assured that when a transaction executes there is no concurrent conflicting transaction being executed remotely and therefore its success depends entirely on the local database engine. In the latter ones, execution is optimistic, each replica independently executes its locally submitted transactions and only then, just before committing, sites coordinate and check for conflicts between concurrent transactions.

This difference results in multiple and often subtle performance and resiliency trade-offs. Namely, how does each protocol cope with a large share of update transactions, conflicting updates, high latency in wide area networks, and symmetric load to multiple replicas. Unfortunately, each protocol is presented using a different load scenario and evaluation method which makes it very hard to clearly highlight the main consequences of the approach.

In this paper we evaluate the suitability of group based replication protocols for replication of On-Line Transaction Processing (OLTP) applications in clusters of servers and over wide area networks (WAN). This evaluation compares the protocols using a common infrastructure which rests on a novel common database interface suitable for the implementation of group based replication protocols. Using the same settings for all protocols and the workload of the industry standard TPC-C benchmark [23] it is possible to establish relative strengths and select the best protocol for each scenario.

The rest of the paper is structured as follows. In Sect. 2, we briefly review the main group-based database replication approaches. Section 3 introduces the common implementation and evaluation framework. Section 4 presents and discusses performance measurements. Finally, Sect. 5 concludes the paper.

2 Replication Protocols

In this study, we consider three replication protocols: CONS, a protocol that implements the conservative execution approach (similar to those proposed in [14, 15]) and two protocols that exploit optimistic execution, Postgres-R (PGR) [13] and the Database State Machine (DBSM) [17]. All of these protocols are multi-master, transactions can be submitted to and executed by several replicas, and follow the passive replication paradigm [4, 8], each transaction is executed by one of the replicas and its state changes propagated to the other replicas.

At the core of these protocols is a total order (or atomic) multicast primitive [10]. Some of the proposed algorithms [16, 14, 15, 20] have been presented using total order primitives with optimistic delivery. The goal is to compensate the inherent ordering latency by allowing tentative processing in parallel with the ordering protocol. If the final order of the messages matches the predicted order then the replication protocol can proceed, otherwise the results obtained tentatively are discarded. We opted, however, not to consider such optimization. In a local area network (LAN), the small message delays discourage any optimistic processing and, in WAN, an algorithm such as the one presented in [21] is required to compensate the large differences and variability of point-to-point latencies. The use of such an algorithm would evenly benefit all of the replication protocols under study but would not contribute to expose the key factors that differentiate them.

The database engine considered implements a multi-version concurrency control mechanism [3]. While locally, the database engine does not provide serializability as its correctness criterion, globally the replication protocols under study are able to do so. In our tests, we will consider and compare both the global 1-copy-serializability [3] and the snapshot-isolation [2] versions of the protocols.

Only update transactions are handled by the replication protocol. Queries are simply executed locally at the database to which they are submitted and do not require any distributed coordination.

In the following, we describe the conservative and optimistic execution approaches and the required interfaces with the database engine.

2.1 Conservative Execution

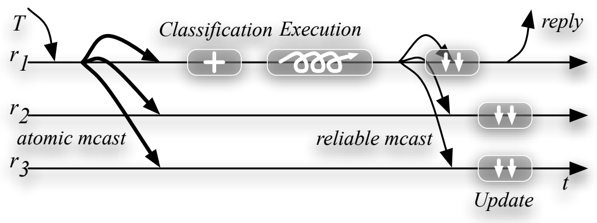


Fig. 1. Conservative replication protocols: CONS

In the conservative approach, data is a priori partitioned in conflict classes, not necessarily disjoint. Each transaction has an associated set of conflict classes (the data partitions it accesses) which are assumed to be known in advance. While the conflict classes for a transaction could be determined at runtime, this would require to know the whole transaction before its execution precluding the processing of interactive transactions.

When a transaction is submitted (Figure 1), its id and conflict classes are atomically multicast to all replicas obtaining a total order position. Each replica has a queue associated with each conflict class and, once delivered, a transaction is classified according to its conflict classes and enqueued in all corresponding queues. As soon as a transaction reaches the head of all of its conflict class queues it is executed. Transactions are executed by the replica to which they are submitted.[1]

Conflicting transactions are executed sequentially. Clearly, the conflict classes have a direct impact on the performance. The fewer the number of transactions with overlapping conflict classes, the better the interleave among transactions. As we shall discuss in Sect. 2.3, conflict classes are usually defined at the table level but can have a finer

[1] When isolated conflict classes exist, dedicating a distinguished replica to the execution of all transactions of such classes, results in a faster processing of those transactions [15].

grain at the expense of a non-trivial validation process to ensure that a transaction does not access conflict classes that were not previously specified.

When the commit request is received, the outcome of the transaction is reliably multicast to all replicas along with the replica's state changes and a reply is sent to the client. Each replica applies the remote transaction's updates with the parallelism allowed by the initially established total order of the transaction.

It is worth noting that, despite the use of a multi-version database engine, since conflicting transactions are totally ordered and executed sequentially, the protocol ensures 1-copy-serializability as long as transactions are classified by the application taking into account read/write conflicts. Relaxing the correctness criterion to snapshot-isolation would simple require the reclassification of the transactions taking into account just write/write conflicts.

2.2 Optimistic Execution

In the optimistic approach, transactions are immediately executed by the replicas to which they are submitted without any a priori coordination. Locally, transactions are synchronized according to the specific concurrency control mechanism of the database engine.

Upon receiving the commit request, a successful transaction is not readily committed. Instead, the tuples read (read-set) and written (write-set) are gathered and a termination protocol initiated. The goal of the termination protocol is to decide the order and the outcome of the transaction such that the global correctness criteria is satisfied. This is achieved by establishing a total order position for the transaction and certifying it (i.e., checking for conflicts) against concurrently executed transactions. The certification of a transaction is done by evaluating the intersection of its read-set and write-set (or just write-set in case of the snapshot-isolation criterion) with the write-sets of concurrent, previously ordered transactions.[2] The fate of a transaction is therefore determined by the termination protocol and a transaction that would locally commit may end up aborted.

The two optimistic protocols, PGR and DBSM (Figure 2), ensure global serializability, but differ in their termination protocols. Both use the transaction's read-sets for the certification procedure. Basically, in PGR the transaction's read-set is not propagated and thus only the replica executing the transaction is able to certify it. In the DBSM, the transaction's read-set is propagated allowing each replica to autonomously certify the transaction.

In detail, upon the reception of the commit request for a transaction t, in PGR the executing replica atomically multicasts t's id and t's write-set and write-values (the values of the tuples in the write-set). As soon as t is ordered, the executing replica certifies t and reliably multicasts the outcome to all replicas. The certification procedure consists in checking t's read-set and write-set against the write-sets of all transactions committed locally since t's commit request.[3] The executing replica then commits or aborts t

[2] The formal definition and detailed explanation of the certification procedures can be found in [13, 16, 24].

[3] In the original protocol [13], a locking concurrency control mechanism was considered for the database engine which allowed to carry the certification process inside the database as part of

locally and replies to the client. Upon the reception of the remote transaction's commit outcome each replica applies t's state changes through the execution of a *high priority* transaction consisting of updates, inserts and deletes according to t's previously multicast write-values. The high priority of the transaction means that it must be assured of acquiring all the required write locks, possibly aborting any locally executing transactions.

The termination protocol in the DBSM is significantly different and works as follows. Upon the reception of the commit request for a transaction t, the executing replica atomically multicasts t's id, the version of the database on which t was executed,[4] and t's read-set, write-set and write-values. As soon as t is ordered, each replica is able to certify t on its own.

For the certification procedure, in the DBSM each replica compares its database version with that of t: if they match t commits, otherwise t's read-set and write-set are checked against the write-sets of all transactions committed locally since t's database version. If they do not intersect, t commits, otherwise t aborts. If t commits then its state changes are applied through the execution of a high priority transaction consisting of updates, inserts and deletes according to t's previously multicast write-values. Again, the high priority of the transaction means that it must be assured of acquiring all the required write locks, possibly aborting any locally executing transactions. The executing replica replies to the client at the end of the transaction.

Of particular relevance for the performance of these two protocols is the definition and representation of the transaction's read-sets. First, read-sets determine the outcome of a transaction certification. If the considered read-set is larger than the set of tuples actually read by the transaction then spurious aborts may arise. On the contrary, if the read-set does not contain the tuples actually read, then serializability may be compromised. Second, in the DBSM protocol the size of the read set may have a serious impact on the network bandwidth. PGR avoids the propagation of the transaction's read-set at the expense of an additional communication step.

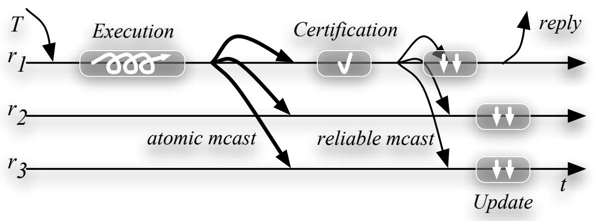
When considering the snapshot-isolation correctness criterion, then both protocols can be simplified and end up being the same. To satisfy snapshot-isolation, certification does not need to check read-write conflicts and thus the transactions' read-sets are not required. As such, the PGR protocol can be simplified by enabling a simpler write-write certification at all the replicas and eliminating the second communication step conveying the outcome of the transaction [24]. The DBSM protocol can also be simplified by not propagating the read-sets and using the simpler certification procedure.

2.3 Database Interface

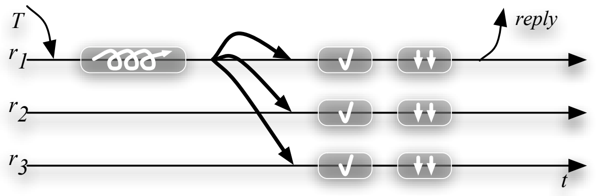
The replication protocols just described require specific interactions with the adopted database engine. Despite their differences, their interaction with the database engine is similar and the interface can be generalized.

the normal execution of the transaction. The read-set was not actually extracted and it consisted of the read locks granted to the transaction.

[4] The database version is a counter maintained by the replication protocol that is incremented every time a transaction commits.



(a) PGR



(b) DBSM

Fig. 2. Optimistic replication protocols

Transactions are submitted to the database engine and evolve through three different phases (Figure 3): the *pre-execute* phase which includes the “begin transaction” command and extends up to the transaction’s first statement, the *execute* phase encompassing the whole transaction execution up to the “commit transaction” command, and the *commit* phase from the "commit transaction" command until the reply to the client application. Interactions between the database engine and the replication protocol happen between these three phases and require extended functionality from the database engine.

In the CONS protocol, at the pre-execute phase the database engine needs to be informed about the conflict classes of the transaction. Usually, such classes are defined at the table level to ease the validation process that occurs at the execution phase to ensure that no other classes beyond those specified at the pre-execute phase are accessed. For finer grains, the process would be more complicated. If instead of whole tables, conflict classes were defined using table partitioning such as filters over attributes, guaranteeing that the accessed items are a sub-set of the conflict classes would ultimately lead to a satisfiability problem [9].

In the optimistic protocols, just before entering the execution phase, a remote transaction is assigned high priority allowing it to break any locks currently granted to other transactions. This interaction is required to ensure the successful execution of the updates of remote transactions. Thus, the concurrency control mechanism of the database engine needs to be extended to distinguish these high priority transactions.

After the local execution of the transaction, before the commit phase, the database engine is required to provide the read-set, write-set and write-values (RWV sets). The write-set and write-values are easily extracted from any database engine but the extrac-

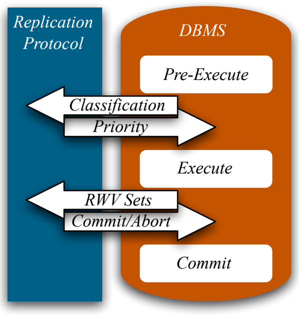


Fig. 3. Interface between the replication protocol and the database engine

tion of the read-set requires close coupling with it. While for simple SPJ statements (i.e., statements that involve Select, Project and Join operations) one extraction step is sufficient, more complex queries require the analysis of the optimizers execution plan, multiple extraction points and further read sets combination. Both PGR and the DBSM protocols rely on the transaction's read-set. A judicious characterization and extraction of the read-set is due to avoid unnecessarily large read-sets and consequent spurious aborts, and to reduce network consumption in the case of the DBSM protocol.

Finally, while naturally the outcome of a transaction is decided inside the database engine, with the optimistic replication protocols the fate of a local transaction ultimately depends on the certification procedure. Therefore, it is required that the database engine allow the replication protocol to determine the commit or abort of the transaction.

With respect to the implementation of the necessary functionality of the database engine most of it needs to be done in core. While one could be tempted to implement these interfaces using a middleware approach through the use of triggers, some, such as the pre-execute and commit interfaces, are not possible with current database engines, and others, such as the extraction of the read-sets would lead to unbearable performance hits.

3 Experimental Procedure

This section presents the simulation environment used to evaluate the protocols. We use a centralized simulation model that combines real software components with simulated hardware, software and environment components to model a distributed system. This allows us to setup and run multiple realistic tests with slight variations of configuration parameters that would otherwise be impractical to perform, specially if one considers a large number of replicas and wide-area networks. The centralized nature of the system allows for global observation of distributed computations with minimal intrusion as well as for control and manipulation of the experiment. All tests are conducted under an implementation that mimics the industry standard on-line transaction processing benchmark TPC-C [23].

3.1 Simulation Infrastructure

To evaluate the protocols we use a hybrid simulation environment that combines simulated and real components [22]. The key components, the replication and the group communication protocols, are real implementations while both the database engine and the network are simulated.

In detail, we use a centralized simulation runtime based on the standard Scalable Simulation Framework (SSF) [6], which provides a simple yet effective infrastructure for discrete-event simulation. Simulation models are built as libraries that can be reused. This is the case of the SSFNet [7] framework, which models network components (e.g. network interface cards and links), operating system components (e.g. protocol stacks), and applications (e.g. traffic analyzers). Complex network models can be configured using these components, mimicking existing networks or exploring particularly large or interesting topologies.

To combine the simulated components with the real implementations the execution of the real software components is timed with a profiling timer [18] and the result is used to mark the simulated CPU busy during the corresponding period, thus preventing other jobs, real or simulated, to be attributed simultaneously to the same CPU. The simulated components are configured according to the equipment and scenarios chosen for testing (Sect. 3.2).

The database server handles multiple clients and is modeled as a scheduler and a collection of resources, such as storage and CPUs, and a concurrency control module. The database provides the interfaces described in Sect. 2.3 (Fig. 3) and implements multi-version concurrency control.

Each transaction is modeled as a sequence of operations: *i*) fetch a data item; *ii*) do some processing; *iii*) write back a data item. Upon receiving a transaction request each operation is scheduled to execute on the corresponding resource. The processing time of each operation is previously obtained by profiling a real database server (Sect. 3.2).

A database client is attached to a database server and produces a stream of transaction requests. After each request is issued, the client blocks until the server replies, thus modeling a single threaded client process. After receiving a reply, the client is then paused for some amount of time (thinking time) before issuing the next transaction request.

To determine the read-set and write-set of a transaction's execution, the database is modeled as a set of histograms [5]. The transactions' statements are executed against this model and the read-set, write-set and write-values are extracted to build the transaction model that is injected into the database server. In our case, this modeling is rather straightforward as the database is very well defined by the TPC-C [23] workload that we use for all tests. Moreover, as all the transactions specified by TPC-C can be reduced to SPJ queries, the read-set extraction is quite simple.

3.2 Test Parameters

Each database request is generated according to the TPC-C benchmark [23]. TPC-C is the industry standard on-line transaction processing benchmark. It mimics a wholesale supplier with a number of geographically distributed sales districts and associated warehouses. TPC-C specifies a precise set of relations (Warehouse, District, Customer, Item, Stock, Orders, OrderLine, NewOrder and History) and the size of the database as a function of the number of desired clients. The benchmark determines 10 clients per warehouse and, as an example, for 2000 clients, the database contains around 10^9 tuples, each ranging from 8 to 655 bytes. The traffic is a mixture of read-only and update intensive transactions. A client can request transactions of five different types: *NewOrder*, adds a new order into the system (with 44% of the occurrences); *Payment*, updates the customer's balance, district and warehouse statistics (44%); *OrderStatus*, returns a given customer latest order (4%); *Delivery*, records the delivery of products (4%); *StockLevel*, determines the number of recently sold items that have a stock level below a specified threshold (4%). The *NewOrder*, *Payment* and *Delivery* are update transactions while the others are read-only.

The database model has been configured using the transactions' processing time of a profiled version of PostgreSQL 7.4.6 under the TPC-C workload. From the TPC-C benchmark we only use the specified workload, the constraints on throughput, performance, screen load and background execution of transactions are not taken into account.

We consider a LAN and a WAN scenarios, both with 9 replicas. In the LAN configuration the replicas are connected by a network with a bandwidth of 1Gbps and a latency of 120μs. The WAN configuration consists of 3 LANs (with 1Gbps and 120μs as before) each with 3 replicas, connected by a network with a bandwidth of 100Mbps and a latency of 60ms. Each replica corresponds to a dual processor AMD Opteron at 2.4GHz with 4GB of memory, running the Linux Fedora Core 3 Distribution with kernel version 2.6.10. For storage we used a fiber-channel attached box with 4, 36GB SCSI disks in a RAID-5 configuration and the Ext3 file system.

For all the experiments that follow, we varied the total of clients from 270 to 3960 and distributed them evenly among the replicas.

4 Experimental Results

4.1 Simple Configuration

The first scenario evaluates the conservative and the DBSM approaches without exploiting any application specific details and thus in a configuration that can easily be automated. In the conservative approach, we use the simple definition of a conflict class for each table, which can actually be easily extracted from the SQL code. The resulting conflict classes and conflict relations among transactions types are shown in the "Serializable" column of Table 1. Regarding the DBSM, we need to pay special attention to read-set sizes since the propagation of large read-sets may be impractical. An immediate workaround to this problem is to set a limit for the read-set size over which the whole table is used. In the TPC-C, this results in transactions of type *Delivery* always being marked as reading the entire *OrderLine* table. All others access only a small number of items.

Figure 4 presents performance measurements in the LAN scenario. It can be observed in Fig. 4(a) that the DBSM protocol with optimistic execution apparently scales much better to a large number of clients than the conservative protocol. As shown by

	Serializable			Snapshot Isolation Level		
Class./Trans	New Order	Payment	Delivery	New Order	Payment	Delivery
Warehouse	x	x			x	
District	x	x		x	x	
Customer	x	x	x		x	x
Item	x					
Stock	x			x		
Orders	x		x	x		x
OrderLine	x		x	x		x
NewOrder	x		x	x		x
History		x			x	

Table 1. Definition of conflict classes for each transaction type in TPC-C.

Fig. 4(b) the bottleneck in the conservative protocol translates in very large queueing latencies.

However, as seen in Fig. 4(c), the good throughput of the DBSM is achieved at the expense of a number of aborted transactions. This is especially worrisome since the 4% of transactions being aborted overall are in fact all *Delivery* transactions as shown in Fig. 4(d). Therefore, even if such transactions can be resubmitted, there is a very low probability of ever being executed. These results show that neither of the approaches scale to a large number of clients with an OLTP load, even with plenty of resources in a LAN.

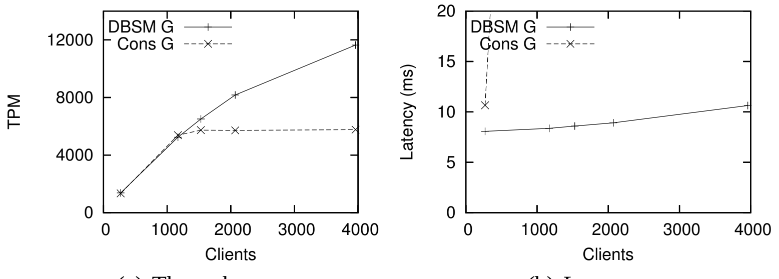
4.2 Fine Granularity

To reduce the number of conflicts, we resort to a finer granularity when defining conflict classes for the conservative approach and the read-set extraction in the DBSM. Fine grained conflict classes are obtained by taking advantage of the fact that all tables except *Item* have references to the *Warehouse* table and that clients connected to the same node have high locality regarding a specific subset of warehouses.

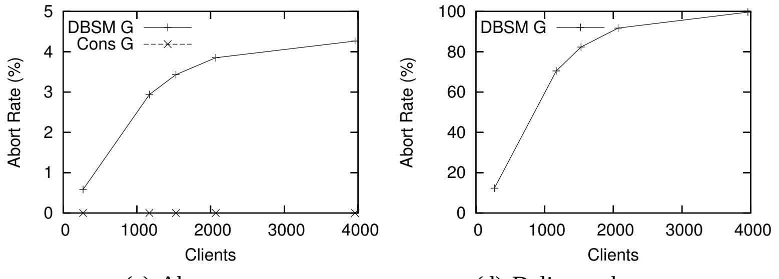
Although this may seem to easily translate in a definition of conflict classes for the conservative protocol, in practice it is not possible because transactions *Payment* and *NewOrder*, which account for a large majority of traffic, may access multiple warehouses. Despite the suitability of this assumption to the TPC-C workload, it must not be generalized since most of the time one cannot be certain of which subset of a table a transaction will access, rendering the approach impractical.

In the optimistic protocol, one uses the same observation to avoid a huge read set without escalating to table level by using only the warehouse attribute and then encoding it as part of the table identifier.

We compare also these optimizations with the PGR protocol which can use the exact read-set by centralizing certification of each transaction. The results are presented in Fig. 5. It can be observed that all approaches produce approximate results with minimal differences in latency and abort rate. Network usage is also very close, showing that the overhead incurred by the DBSM when sending the read-set is offset by requiring only a single communication step. These results show that with an appropriate granularity, all



(a) Throughput (b) Latency



(c) Abort rate (d) Delivery abort rate

Fig. 4. Performance measurements in a LAN with coarse granularity.

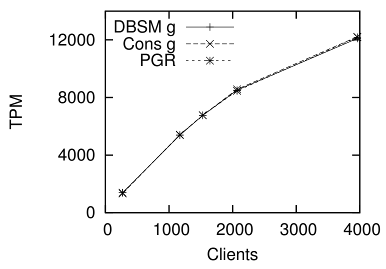
these group communication based protocols are equally appropriate for an OLTP load in a cluster.

4.3 Snapshot Isolation

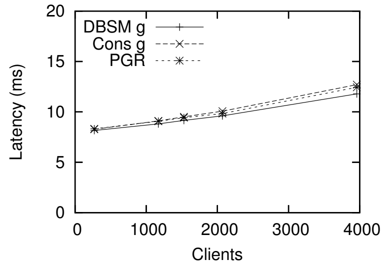
An alternative approach to avoid synchronization conflicts is to relax the correctness criterion to snapshot isolation [2] which only considers write-write conflicts.

In the DBSM approach, all the concerns previously discussed about the size of the read-set are avoided. As Fig. 6 shows, it turns out that this alternative has also a benign impact on the performance of the DBSM approach, reducing the number of aborted transactions. Moreover, this is a very appealing alternative, as it avoids all configuration issues. As explained in Sect. 2.2, under snapshot isolation the DBSM and PGR protocols become the same.

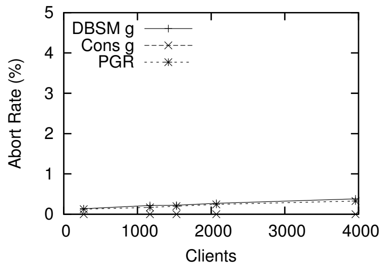
Unlike the DBSM, the conservative approach does not benefit from the snapshot isolation criterion, exhibiting the same latency as before. In the "Snapshot Isolation Level" column of Table 1 the new conflict relations among the transactions are depicted. Regardless of their type, all update transactions still conflict and thus have to be sequentially executed.



(a) Throughput



(b) Latency



(c) Abort rate

Fig. 5. Performance measurements in a LAN with fine granularity.

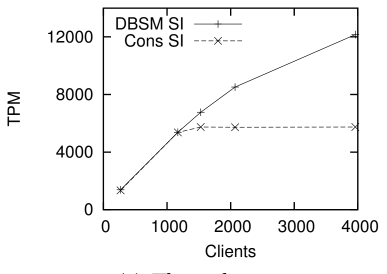
4.4 Wide Area

Finally, we are interested in observing how the proposed approaches scale also to inter-connected clusters in WAN. The best performers in the previous scenarios were chosen and their performance in this environment is presented Fig. 7. Although Fig. 7(a) shows that throughput scales equally well, Fig. 7(b) shows that the additional communication step, incurred by PGR, when centralizing certification results in a large increase in latency. This has also an impact in the overall abort rate in Fig. 7(c), which is higher than with other optimistic approaches. Note however that, in contrast with the results of Fig. 4, Fig. 7(d) shows that no single transaction type exhibits high abort rates, hence, if one chooses to resubmit the aborted transactions there is a high probability of a successful execution.

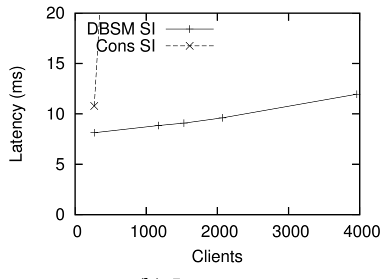
4.5 Discussion

The key issue in obtaining close to linear scalability of a distributed system is reducing synchronization overhead. In practice, one can measure this overhead by the time the computation in a node is suspended waiting for interaction with remote nodes. In a traditional protocol based on distributed locking, this can potentially be very large, if a node has to wait that all other nodes enter and leave a critical section plus the time it takes to pass the authorization around.

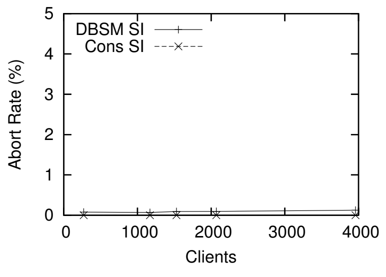
In contrast, when using active replication [19, 8], the only overhead is encapsulated in the total order multicast protocol and no additional synchronization is required. Ide-



(a) Throughput



(b) Latency



(c) Abort rate

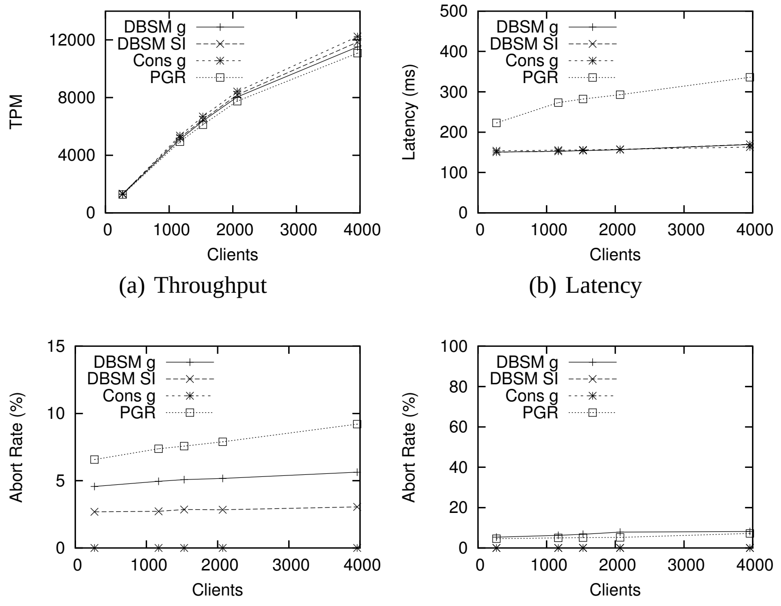
Fig. 6. Performance measurements in a LAN with snapshot isolation.

ally, a database replication protocol based on total order multicast would be able to achieve the same goal. We now examine in turn each of the protocols to determine how this goal is achieved.

Figure 8 depicts the conservative and optimistic protocols handling the execution of two concurrent non-conflicting transactions. In the CONS protocol (Fig. 8(a)), once the transactions are ordered all steps of the protocol are executed concurrently therefore corresponding to the desired behavior.

Regarding the optimistic approaches, we can see that in the DBSM (Fig. 8(b)) the transactions' execution can always be carried in parallel while the certification procedure needs to be done sequentially. Once the certification is finished, since the transactions do not conflict, the updates may be incorporated concurrently. The DBSM therefore incurs in the certification procedure overhead. However, the certification execution time is usually negligible though.

In contrast, the PGR protocol is penalized by the supplemental reliable multicast. Although the transactions' execution can be done in parallel too, the certification of T' (ordered after T) can only be done once r_3 knows the outcome of T. That is, the latency of the reliable multicast of T is incorporated in the response time of T'. This problem can further suffer a cascading effect caused by the expected system parallelism.



(a) Throughput (b) Latency

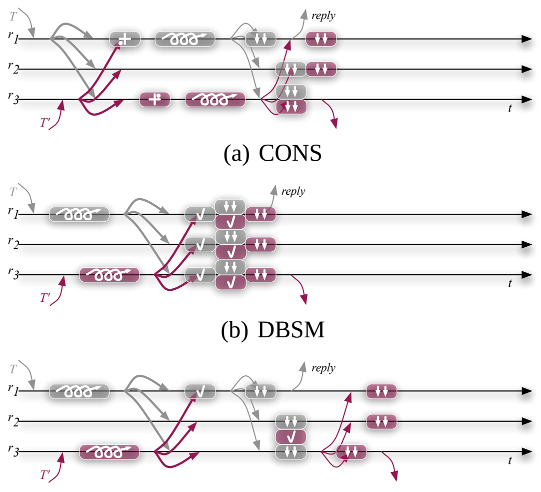
(c) Abort rate (d) Delivery abort rate

Fig. 7. Performance measurements in a WAN.

5 Conclusions

Database replication protocols based on group communication have been previously evaluated with a variety of implementation or simulation techniques and also a variety of, often non-representative, loads or system models. When an industry standard database benchmark is used, it is often TPC-W, which provides a read-intensive load which does not stress synchronization mechanisms. This makes it difficult to compare their relative trade-offs and performance regarding non-replicated databases.

In contrast, in this paper, we use the realistic write-intensive OLTP load from the TPC-C benchmark and in Sect. 4 we show that high performance and close to linear scalability can be achieved with several configurations. In detail, we show that when snapshot isolation suffices for the application requirements, as happens with TPC-C itself, the DBSM-SI protocol is the best option, requiring little effort to configure and offering excellent performance in LAN and WAN. When serializability is required, there are two possible options. When adequately fine grained conflict classes can be defined, predicted beforehand for each transaction, and the source modified to convey them, a conservative protocol provides excellent performance without introducing transaction aborts. If adequately fine-grained conflict classes cannot be predicted or the source modified to tag transactions or cope with snapshot isolation, as happens when supporting large third party legacy applications, the DBSM protocol provides the same performance when given an adequate definition of read-set granularity. Notice that this



(a) CONS

(b) DBSM

(c) PGR

Fig. 8. Handling concurrent transactions

can be achieved by a database administrator with no modification of sources and with no impact on correctness, which provides maximum flexibility and safety.

In short, group communication based database replication protocols provide a spectrum of configurability, generality and performance trade-offs that fit the most demanding applications. The wide availability of such protocols therefore demands improved database interfaces that efficiently provide the functionality identified in Sect. 2 of this paper.

References

[1] D. Agrawal, A. El Abbadi, and R. C. Steinke. Epidemic algorithms in replicated databases (extended abstract). In *Proc. ACM Symp. Principles of Database Systems (PODS)*, 1997.

[2] H. Berenson, P. Bernstein, J. Gray, J. Melton, E. O'Neil, and P. O'Neil. A critique of ANSI SQL isolation levels, 1995.

[3] P. Bernstein, V. Hadzilacos, and N. Goodman. *Concurrency Control and Recovery in Database Systems*. Addison-Wesley, 1987.

[4] N. Budhiraja, K. Marzullo, F. Schneider, and S. Toueg. The primary-backup approach. In S. Mullender, editor, *Distributed Systems*, chapter 8. Addison Wesley, 1993.

[5] A. Correia, A. Menezes, and R. Oliveira. Off-line test automation for database replication based on group communication. Technical report, Universidade do Minho, 2005.

[6] J. Cowie. *Scalable Simulation Framework API Reference Manual*, March 1999.

[7] J. Cowie, H. Liu, J. Liu, D. Nicol, and Andy Ogielski. Towards realistic million-node internet simulation. In *Proc. Int'l Conf. Parallel and Distributed Processing Techniques and Applications (PDPTA'99)*, 1999.

[8] R. Guerraoui and A. Schiper. Software-based replication for fault tolerance. *IEEE Computer*, 30(4), April 1997.

[9] S. Guo, W. Sun, and M. Weiss. Solving Satisfiability and Implication Problems in Database Systems. *ACM Transactions on Database Systems (TODS)*, 1996.

[10] V. Hadzilacos and S. Toueg. A modular approach to fault-tolerant broadcasts and related problems. Technical Report TR94-1425, Cornell Univ., Computer Science Dept., May 1994.

[11] J. Holliday, D. Agrawal, and A. El Abbadi. The performance of database replication with group multicast. In *Proc. IEEE Int'l Symp. Fault-Tolerant Computing Systems (FTCS)*, 1999.

[12] B. Kemme and G. Alonso. A suite of database replication protocols based on communication primitives. In *Proc. IEEE Int'l Conf. Distributed Computing Systems (ICDCS)*, 1998.

[13] B. Kemme and G. Alonso. Don't be lazy, be consistent: Postgres-r, a new way to implement database replication. In *VLDB '00: Proceedings of the 26th International Conference on Very Large Data Bases*, pages 134–143, San Francisco, CA, USA, 2000. Morgan Kaufmann Publishers Inc.

[14] B. Kemme, F. Pedone, G. Alonso, and A. Schiper. Processing transactions over optimistic atomic broadcast protocols. In *Proc. IEEE Int'l Conf. Distributed Computing Systems (ICDCS)*, 1999.

[15] M. Patiño-Martínez, R. Jiménez-Peris, B. Kemme, and G. Alonso. Scalable replication in database clusters. In *DISC'00: Proceedings of the 14th International Conference on Distributed Computing*, pages 315–329, London, UK, 2000. Springer-Verlag.

[16] F. Pedone. *The Database State Machine and Group Communication Issues*. PhD thesis, Département d'Informatique, École Polytechnique Fédérale de Lausanne, 1999.

[17] F. Pedone, R. Guerraoui, and A. Schiper. The database state machine approach. *J. Distributed and Parallel Databases and Technology*, 2003.

[18] M. Pettersson. Linux performance counters. http://user.it.uu.se/ mikpe/linux/perfctr/, 2004.

[19] F. Schneider. Replication management using the state-machine approach. In S. Mullender, editor, *Distributed Systems*, chapter 7. Addison Wesley, 1993.

[20] A. Sousa, F. Pedone, R. Oliveira, and F. Moura. Partial replication in the database state machine. In *IEEE Int'l Symp. Networking Computing and Applications*, 2001.

[21] A. Sousa, J. Pereira, F. Moura, and R. Oliveira. Optimistic total order in wide area networks. In *Proc. IEEE Int'l Symp. Reliable Distributed Systems (SRDS)*, 2002.

[22] A. Sousa, J. Pereira, L. Soares, A. Correia Jr., L. Rocha, R. Oliveira, and F. Moura. Testing the dependability and performance of group communication based database replication protocols. In *IEEE Intl. Conf. on Dependable Systems and Networks - Performance and Dependability Symposium (DSN-PDS'2005)*, 2005. to appear.

[23] Transaction Processing Performance Council (TPC). TPC Benchmark^{TM} C standard specification revision 5.0, February 2001.

[24] S. Wu and B. Kemme. Postgres-r(si): Combining replica control with concurrency control based on snapshot isolation. In *Proc. of the IEEE Int. Conf. on Data Engineering (ICDE)*, pages 422–433, April 2005.